VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

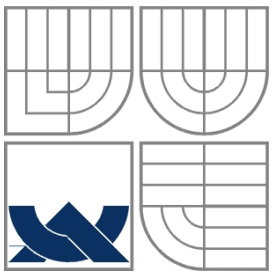
HARDWAROVÁ AKCELERACE FILTRACE OBRAZU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

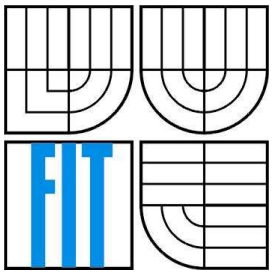
AUTOR PRÁCE
AUTHOR

MARTIN ZELINKA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE FILTRACE OBRAZU

HARDWARE-BASED ACCELERATION OF IMAGE FILTRATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ZELINKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2009

Zadání bakalářské práce/8954/2008/xzelin12

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok

Zadání bakalářské práce

Řešitel: **Zelinka Martin**
Obor: Informační technologie
Téma: **Hardwarová akcelerace filtrace obrazu**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s problematikou filtrace obrazu pomocí FIR filtrů a technikami jejich hardwarové akcelerace.
2. Využijte vhodnou FPGA platformu a navrhněte systém, který bude schopen demonstrovat akceleraci filtrace. Systém musí umožňovat snadnou změnu koeficientů FIR filtru.
3. Navržený systém implementujte.
4. Ověřte funkčnost na sadě testovacích dat a několika typech FIR filtrů. Vyhodnoťte dosažené zrychlení.

Literatura:

- Ramesh, J.: *Machine Vision*. Boston, McGraw-Hill, 1995.
- Smith, S. W.: *Digital Signal Processing*. California Technical Publishing, San Diego, 1999.
- Meyer-Baese, U.: *Digital signal processing with field programmable gate arrays*, Springer,2004.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vašíček Zdeněk, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá hardwarovou akcelerací filtrace obrazu s využitím FIR filtrů. Definuje základní pojmy týkající se digitálního obrazu, popisuje princip filtrace a stručně vysvětluje techniky používané při detekci hran v obraze a při vyhlazování obrazu. Hlavním cílem práce je rozbor několika metod akcelerace FIR filtrů, které jsou vhodné pro realizaci v hardwaru, a následná implementace vybrané metody s možností změny konfigurace za běhu s ohledem na maximální propustnost. V závěru práce je uvedeno vyhodnocení metody z hlediska propustnosti a provedeno srovnání s optimální softwarovou implementací.

Abstract

This bachelor's thesis deals with the hardware-based acceleration of image filtration using FIR filters. Define basic concept about digital image, describes the principle of filtration and briefly explains the techniques used to detect edges in an image and smoothing the image. The main aim of this work is the analysis of several acceleration methods of FIR filters, which are suitable for implementation in hardware, and the subsequent implementation of the selected method with the possibility of configuration changes at runtime with regard to the maximum throughput. In conclusion of this work it is described the evaluation of the method and made compared with the optimal software implementations.

Klíčová slova

2D FIR filtr, filtrace obrazu, hardwarová akcelerace, distribuovaná aritmetika, RAG algoritmus, FPGA

Keywords

2D FIR filter, image filtration, hardware-based acceleration, distributed arithmetic, RAG algorithm, FPGA

Citace

Zelinka Martin: Hardwarová akcelerace filtrace obrazu, bakalářská práce, Brno, FIT VUT v Brně, 2009

Hardwarová akcelerace filtrace obrazu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka.

……
Martin Zelinka
20. května 2009

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Ing. Zdeňkovi Vašíčkovi za odborné vedení a čas věnovaný konzultacím této práce.

Obsah

1 Úvod

Zpracování digitálního obrazu patří mezi významné části počítačové grafiky. Jedná se především o zpracování digitálních fotografií, jako je například potlačení šumu nebo zvýraznění určitých detailů. Zmíněné operace se nejčastěji provádí pomocí obrazových FIR filtrů, které pracují na principu lineární konvoluce. S postupem doby se ale zvětšuje kvalita, tedy rozlišení fotografií, a rostou nároky na rychlost zpracování obrazu. Rychlost můžeme zvýšit použitím efektivnějších algoritmů nebo realizací filtru v hardwaru. Tato práce se zabývá postupy, které lze uplatnit při implementaci obrazových FIR filtrů v hradlových polích (FPGA). Při použití vhodného návrhu mohou aplikace realizované v FPGA dosahovat vysokých výkonů.V druhé části práce je pak popsána implementace konkrétního obrazového filtru pomocí jedné vybrané metody.

Práce je rozdělena do následujících kapitol. První kapitola seznamuje čtenáře s reprezentací digitálního obrazu v počítači a popisuje princip, na kterém pracují obrazové filtry. Dále popisuje rozdělení filtrů a uvádí několik konkrétních často používaných filtrů. Další kapitola se věnuje hardwarové akceleraci filtrů. Popisuje různé metody, které lze uplatnit při realizaci FIR filtrů v hradlových polích, aby bylo dosaženo co největšího výkonu. Jednotlivé metody jsou vysvětleny na konkrétních příkladech, aby je čtenář snadno pochopil. Následující kapitola popisuje použitý hardware. V první časti seznamuje čtenáře obecně se strukturou FPGA a v druhé části pak popisuje konkrétní vybranou platformu. Další kapitola se věnuje implementaci vybrané metody hardwarové akcelerace. Podrobně seznamuje čtenáře s jednotlivými částmi celé architektury a vysvětluje jejich funkci. Na konci této kapitoly je zmíněna softwarová aplikace zajišťující komunikaci uživatele s hardwarovou architekturou. V poslední kapitole jsou uvedeny dosažené výsledky. Hardwarová realizace je porovnána se softwarovou aplikací, aby se dalo zjistit zrychlení navržené architektury. Na závěr je uvedeno zhodnocení celé práce, zmíněny případná rozšíření a další možné pokračování projektu.

2 Obraz a filtrace obrazu

2.1 Reprezentace obrazu

Obraz může být v počítačové grafice reprezentován dvěmi způsoby. První možností je vektorový obraz a druhou možností je rastrový obraz. Ve vektorovém obrazu se data ukládají ve formě vektorových entit (úsečky, kružnice atd.), zatímco v rastrové grafice jsou data uložena v rastrové matici. Vektorový obraz se používá na reprezentaci např. grafů, písma a hlavně ve 3D grafice, zatímco rastrový obraz se používá na reprezentaci digitálních fotografií (obrazů), a proto si o něm řekneme více.

2.1.1 Rastrový obraz

Rastrový obraz, jak již bylo řečeno, je v podstatě matice bodů, které nesou pouze informaci o své barvě. Neobsahují žádné informace o objektech, které jsou na obraze zobrazeny [2]. Body obrazu se nazývají pixely (z anglických slov picture element, obrazový prvek). Pixel je nejmenší jednotka obrazové informace a lze si ho představit jako čtverec vyplněný barvou. Počtu řádků a sloupců pixelů v celém obraze se říká rozlišení obrazu. Čím větší rozlišení má obraz, tím lépe dokáže zobrazit drobné detaily.

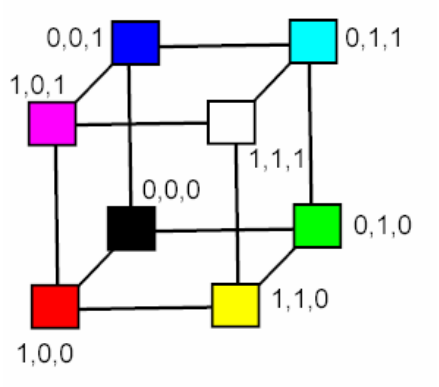
U pixelu se setkáme s termínem barevná hloubka. Barevná hloubka určuje počet bitů potřebných na reprezentaci barvy jednoho pixelu. Pokud máme např. barevný model RGB (viz následující kapitola 2.1.2) a každá barva je zapsána na osmi bitech (tedy 256 různých hodnot), tak barevná hloubka obrazu je 24 bitů (počet barev krát počet bitů na barvu) [3]. S tím souvisí pojem kanál (channel), který udává, z kolika barevných složek se skládá barva jednoho pixelu. U již zmíněného barevného modelu RGB jsou definovány 3 kanály (červený = red, zelený = green a modrý = blue).

2.1.2 RGB barevný model

V počítačové grafice může být barva reprezentována různými způsoby. Podle toho máme různé barevné modely (RGB, CMY, HLS, HSV). Velmi často používaným modelem je RGB model, jehož počátky jsou spojeny s počítačovou technikou. Používá se v zobrazovací technice (monitory, projektory, televizory). Obsahuje tři barevné složky, červenou, zelenou a modrou, a využívá principu, že v lidském oku se nachází světločivné buňky, které reagují právě na tyto tři barvy. Model používá aditivní skládání barev, které je podobné principu skládání barev světla. Při přidání nového odstínu se barva zesvětlí, takže všechny odstíny dohromady tvoří bílou barvu a žádný odstín nám dá černou barvu. Tento princip lze reprezentovat pomocí jednotkové krychle (viz obrázek 2.1), kde v počátku (0, 0, 0) se nachází černá barva a v protilehlém rohu (1, 1, 1) bílá barva. Barevné odstíny vznikají skládáním základních barev, jejichž intenzita se udává v intervalu <0, 1>. V počítačové grafice se používá nejčastěji dělení intervalu na 256 dílků (8 bitů) v rozmezí <0, 255>, což nám udává barevnou hloubku (viz předchozí kapitola 2.1.1) [8].

Pokud mají všechny 3 barevné složky stejnou barvu, jedná se o obraz v odstínech šedi. U barevného modelu RGB se převod z barevného obrazu na obraz v odstínech šedi provádí podle rovnice 2.1. Různé barevné složky mají různé zastoupení ve výsledném obrazu, protože lidské oko je jinak citlivé na každou barevnou složku.

$$I = 0{,}299R + 0{,}587G + 0{,}114B \tag{2.1}$$



Obrázek 2.1: Barevný model RGB [8]

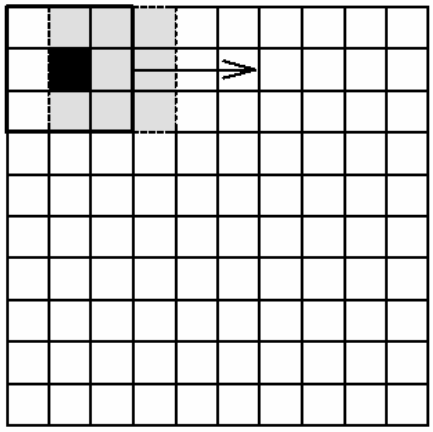
RGB model má ještě druhou variantu, která se označuje RGBA. Každá barva obsahuje navíc alfa kanál, který nese informaci o průhlednosti. Kanál alfa určuje poměr smísení barvy bodu s barvou pozadí. Hodnota 0 udává, že bod je zcela průhledný, zatímco hodnota 1, že je zcela neprůhledný [8].

2.2 Princip filtrace obrazu

Lineární 2D FIR filtry pracují na principu lineární konvoluce, proto jsou někdy označovány jako konvoluční filtry. Výpočet výsledného obrazu probíhá pixel po pixelu , kde pixel výstupního obrazu se spočte jako konvoluce jeho okolí a konvolučního jádra, nebo-li jinak řečeno vynásobením bodu a jeho okolí odpovídajícími hodnotami konvolučního jádra a sečtením těchto hodnot. Konvoluční jádro, označované někdy také jako maska či filtrační okno, je nejčastěji reprezentováno čtvercovou maticí o lichém počtu řádků a sloupců (např. 3×3, 5×5, 7×7) [9]. Dvourozměrnou lineární konvoluci, nebo-li výsledné pixely filtrovaného obrazu, lze popsat vztahem

$$y[k,l] = x[k,l] * h[k,l] = \sum_{i=-\frac{I}{2}}^{\frac{I}{2}} \sum_{j=-\frac{J}{2}}^{\frac{J}{2}} h[i,j]x[k-i,l-j] \tag{2.2}$$

kde x[k, l] jsou pixely vstupního obrazu, y[k, l] pixely výstupního obrazu a h[k, l] konvoluční jádro o velikosti I + 1 × J + 1. Způsob, jakým probíhá konvoluce obrazu, je naznačen na obrázku 2.2.

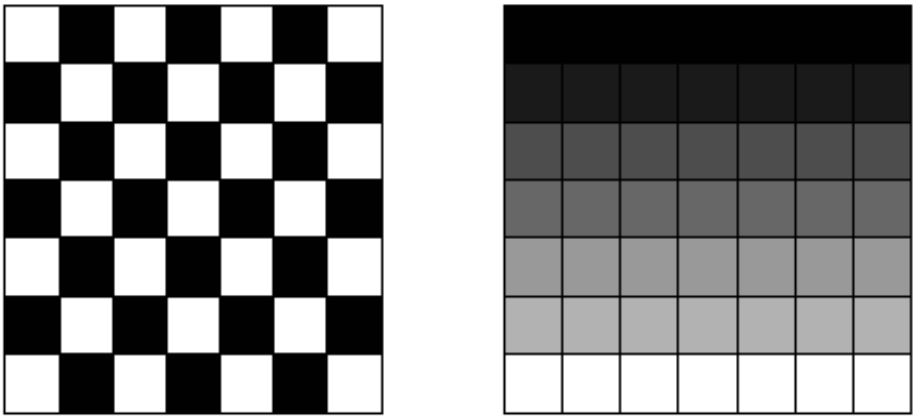


Obrázek 2.2: Princip filtrace obrazu

Aby výsledný pixel byl vždy uprostřed filtračního okna, musíme zvlášť vyřešit situaci pro okrajové body po obvodu obrazu. Nejlepší výsledek dostane, pokud budeme replikovat okrajové pixely. Další možností je, buď doplnit po obvodu obrazu nuly, nebo zmenšit výsledný filtrovaný obraz o polovinu šířky (výšky) filtrovacího okna mínus jedna na všech stranách [9].

Podobně jako jednorozměrný signál obsahuje i obraz nízkofrekvenční a vysokofrekvenční informaci. Vysoké frekvence jsou dány velkým rozdílem hodnot sousedních pixelů a určují tedy hrany obrazu. Nízké frekvence popisují postupné přechody v obrazu a určují tedy velké jednotné plochy. Podle toho jaké frekvence filtr propouští můžeme rozdělit filtry na vysokofrekvenční a nízkofrekvenční.

Kromě lineárních obrazových filtrů existují ještě nelineární obrazové filtry, které se také často využívají při zpracování obrazu. Tyto filtry často využívají lineární filtry pro předzpracování obrazu.



Obrázek 2.3: Vlevo – vysokofrekvenční informace, vpravo – nízkofrekvenční informace [9]

2.2.1 Nízkofrekvenční filtry

Nízkofrekvenční filtry propouští nízké frekvence a potlačují vysoké frekvence. Výsledný obraz je oproti originálu shlazený a neobsahuje ostré hrany. Čím větší masku použijeme, tím dosáhneme většího shlazení. Nízkofrekvenční filtry se používají na rozmazání obrazu a na odstranění šumu z obrazu. Mezi tyto filtry patří například průměrový nebo Gaussův filtr. Dalším nízkofrekvenčním filtrem je např. mediánový filtr, který dosahuje mnohem lepších výsledků při odstranění šumu, ale nepatří mezi lineární FIR filtry, a tak se o něm nebudu více zmiňovat.

2.2.1.1 Průměrový (Mean) filtr

Průměrový filtr pracuje na velmi jednoduchém principu. Počítá průměr svého okolí a sebe, a tak jsou odstraněny pixely, které jsou odlišné od svého okolí [10]. Filtr se tedy často používá na odstranění šumu z obrazu. Čím větší masku použijeme, tím zahrneme do výpočtu větší okolí, a tím bude i výraznější výsledný efekt. Nejčastěji se používá maska o rozměru 3×3 viz obrázek 2.4.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Obrázek 2.4: Filtrační maska průměrového filtru

2.2.1.2 Gaussův filtr

Gaussův filtr pracuje na podobném principu jako průměrový filtr, ale používá masku s Gaussovým rozložením [11]. 2D Gaussovo rozložení můžeme popsat rovnicí 2.3.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.3}$$

kde σ je odchylka.

Na rozdíl od průměrového filtru o stejné velikosti, je výsledek jemněji vyhlazený a zachovává lépe hrany. Nejčastěji se tento filtr používá na odstranění šumu z obrazu a na rozostření. Maska o rozměru 3×3 může vypadat např. viz obrázek 2.5.

$$1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Obrázek 2.5: Filtrační maska Gaussova filtru

2.2.2 Vysokofrekvenční filtry

Vysokofrekvenční filtry propouští vysoké frekvence a potlačují nízké frekvence. Ve výsledném obrazu vystoupí hrany. Vysokofrekvenční filtry se používají na ostření obrazu nebo na zvýraznění hran obrazu. Mezi tyto filtry patří například Laplaceův nebo Sobelův filtr.

2.2.2.1 Laplaceův filtr

Tento filtr se používá na detekci hran v obraze. Metoda je ale náchylná na šum, a tak je dobré na obraz nejprve použít nějakou metodu na odstranění šumu [4]. Dvě používané masky pro čtyřokolí a osmiokolí o rozměrech 3×3 jsou na obrázku 2.6. Váha prostředního koeficientu je rovna součtu vah všech ostatních koeficientů a součet všech koeficientů je roven nule. Filtr se chová tak, že homogenní plochy budou mít nulové hodnoty a pixely s menší nebo větší hodnotou než okolí budou mít nízké respektive vysoké hodnoty [9].

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Obrázek 2.6: Filtrační masky Laplaceova filtru, vlevo – čtyřokolí, vpravo – osmiokolí

2.2.2.2 Sobelův filtr

Sobelův filtr zvýrazňuje buď horizontální nebo vertikální hrany. Případně můžeme oba zkombinovat a získat tzv. celkový Sobelův gradient, pak už se ale jedná o nelineární filtr. První se aplikuje jedna filtrační matice a potom druhá [9]. Výsledný Sobelův gradient se pak spočte podle rovnice 2.4.

$$G = \sqrt{X^2 + Y^2} \tag{2.4}$$

Na stejném principu jako Sobelův filtr pracují i další filtry. Mezi ně patří např. Prewittův, Robertsonův nebo Kirschův filtr.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Obrázek 2.7: Filtrační masky Sobelova filtru na zvýraznění horizontálních (vlevo) a vertikálních hran

2.2.2.3 Ostřící filtr

Ostřící filtry jsou podobné filtrům na zvýraznění hran. Musí u nich ale zůstat zachován jas obrazu, a proto se musí suma hodnot ve filtrační matici rovnat 1. Filtrační matice může vypadat např. jako na obrázku 2.8. První varianta beru v úvahu pouze čtyřokolí bodu a druhá osmiokolí. Druhá varianta dává větší důraz na středový bod a tím může zvýraznit případný šum, což nechceme [4].

0	-0,5	0
-0,5	3	-0,5
0	-0,5	0

-1	-1	-1
-1	9	-1
-1	-1	-1

Obrázek 2.8: Filtrační masky ostřícího filtru, vlevo – čtyřokolí, vpravo - osmiokolí

3 Techniky hardwarové akcelerace FIR filtrů

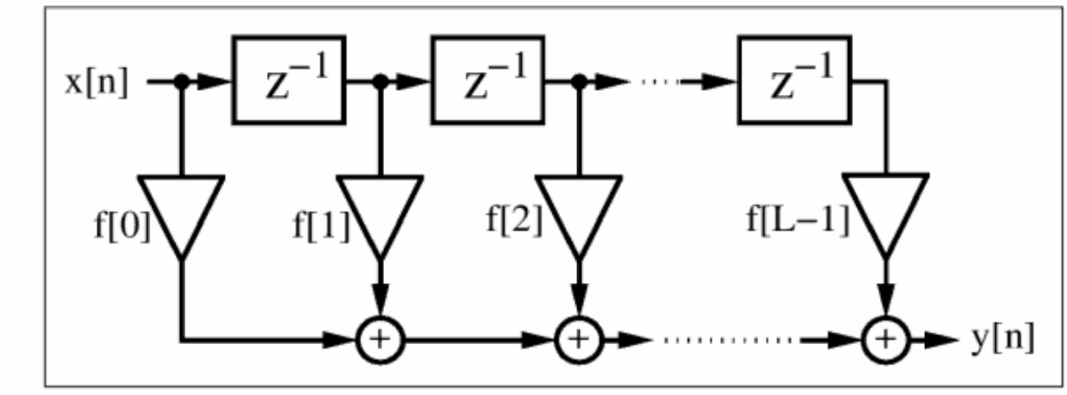
Implementace konkrétní úlohy v hardwaru je mnohokrát rychlejší než implementace v softwaru. Mezi hlavní důvody, proč hardware pracuje rychleji, patří paralelní a zřetězené zpracování. Dalším důvodem je, že hardware se většinou specializuje pouze na jednu konkrétní úlohu, pro kterou byl navržen. Často se používají výpočetní jednotky co nejlépe přizpůsobené konkrétnímu algoritmu. Techniky používané při realizaci FIR filtrů popisuje tato kapitola. Principy jednotlivých metod jsou vysvětleny pro jednoduchost na jednorozměrných FIR filtrech.

3.1 Přímá metoda

Při použití přímé metody obsahuje implementace filtru jednotlivé komponenty, tedy sčítačky, násobičky a zpožďovací členy (registry). Kromě tohoto způsobu můžeme ve VHDL implementovat celý filtr jako jeden proces. Potom necháváme větší práci na syntetizátoru, který případně optimalizuje navržené řešení [1]. Přímá metoda mívá nízkou rychlost a často zabírá hodně velkou plochu na obvodu FPGA, proto se příliš často nepoužívá. Přímou metodu můžeme dále vylepšit pomocí následujících tří úprav [1]:

- Všechny koeficientu filtru zapsat pomocí CSD kódu.
- Zefektivnit násobení pomocí zřetězeného zpracování.
- U filtrů se symetrickými koeficienty můžeme snížit počet násobiček.

Ale i po provedení následujících úprav se rychlost nijak extrémně nezvýší, a proto se používají jiné metody, které se snaží hlavně vyhnout násobení a nahradit ho pouze sčítáním a bitovými posuny.



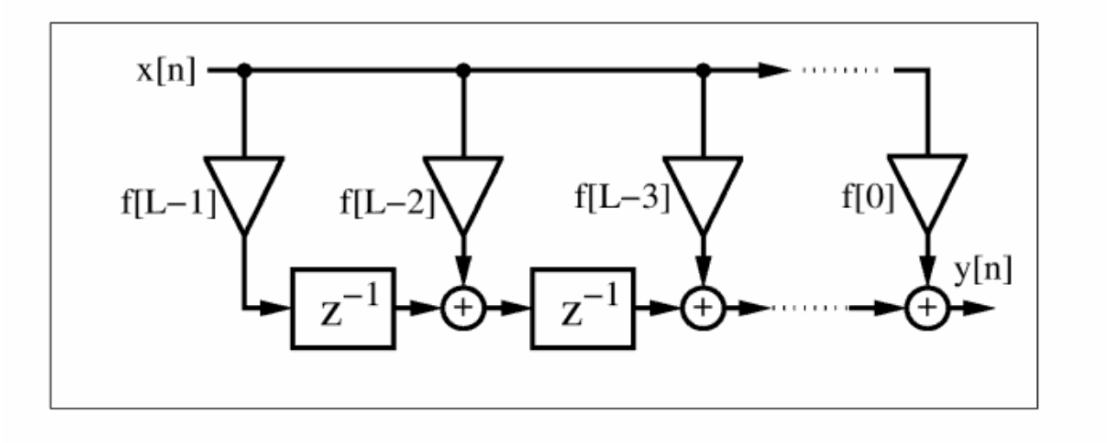
Obrázek 3.1: Přímá forma FIR filtru [1]

3.2 Transponovaná struktura

Jednou z variant implementace FIR filtrů je transponovaná struktura. Vznikne z klasického FIR filtru provedením následujících úprav [1]:

- Prohozením vstupu za výstup.
- Změnou směru toku signálu.
- Nahrazením větvení sčítačkami a naopak.

Tato varianta je lepší a preferovanější než přímá metoda, protože nepotřebujeme navíc posuvný registr pro vstupní vzorky. Navíc můžeme aplikovat další vylepšení jako je násobné využití opakujících se koeficientů pomocí RAG algoritmu (reduce adder graph, snížení počtu sčítaček) nebo použití zřetězených sčítaček s uchováním přenosu. Zřetězené sčítačky by měly zvýšit rychlost, zatímco RAG algoritmus by měl zvýšit rychlost i zmenšit velikost návrhu [1].



Obrázek 3.2: Transponovaná struktura FIR filtru [1]

3.2.1 RAG algoritmus

Algoritmus pracuje na principu, že všechny koeficienty můžeme zapsat jako součet mocnin čísla dvě. Dále využívá předpokladu, že některé koeficienty budou součástí jiných koeficientů. Např. máme-li koeficienty 9 a 11, tak 9 zapíšeme jako 8 + 1 a 11 zapíšeme jako 9 + 2. Tím snížíme počet sčítaček o jednu. Nalezení optimálního počtu sčítaček je tzv. NP-úplná úloha. Postupuje se podle následujících kroků [1]:

1. Odstraníme znaménka, protože znaménka můžeme realizovat pomocí odčítání ve zpožďovací lince filtru.
2. Odstraníme všechny koeficienty a jejich dělitele, které jsou mocninou čísla dvě, protože je můžeme implementovat pomocí bitových posunů.
3. Realizujeme všechny koeficienty, jejichž cena je rovna jedné (jež lze realizovat pomocí jedné sčítačky).
4. Pomocí koeficientů s cenou jedna, realizujeme koeficienty s vyšší cenou.

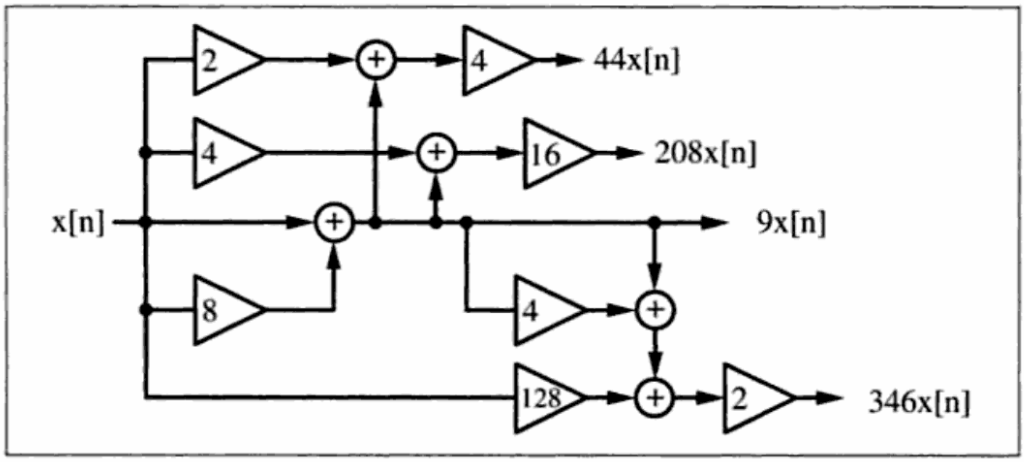
Celý algoritmus nejlépe pochopíme na konkrétním příkladu (podle [1]): Mějme filtr s koeficienty f[k], jejichž hodnoty jsou 346, 208, -44 a 9. Nejprve si určíme kolik sčítaček by bylo potřeba bez použití RAG algoritmu.

f[k]	Cena (počet sčítaček)
f[0] = 346 = $2 \times 173 = 2 \times (128 + 32 + 8 + 4 + 1)$	4
f[1] = 208 = $2^4 \times 13 = 2^4 \times (8 + 4 + 1)$	2
f(2) = -44 = $2^2 \times 11 = 2^2 \times (8 + 2 + 1)$	2
f(3) = 9 = $3^2 = (2 + 1)^2$	1
Celkem	**9**

Celkový počet sčítaček je 9. Nyní na stejný filtr aplikujeme RAG algoritmus.

Krok	Zbývá realizovat	Již realizováno	Akce/nalezená reprezentace
0	346, 208, -44, 9	-	Inicializace
1	346, 208, 44, 9	-	Odstranění znaménka
2	346, 208, 44, 9	-	Odstranění 2^k koeficientů
3	173, 13, 11, 9	-	Odstranění 2^k dělitelů z koeficientů
4	173, 13, 11	9	Realizace koeficientů s cenou 1
5	173, 13, 11	9	Ostatní koeficienty mají cenu vyšší než 1
6	173, 13	9, 11	11 = 9 + 2
7	173	9, 11, 13	13 = 9 + 4
8	-	9, 11, 13, 173	$173 = (4 + 1) \times 9 + 128$

Na obrázku 3.3 vidíme výsledek RAG algoritmu. Počet sčítaček byl snížen z 9 na 5. Zmenšilo se také zpoždění filtru ze 4 na 3.



Obrázek 3.3: Realizace filtru pomocí RAG algoritmu [1]

3.3 Distribuovaná aritmetika

Úplně jinou technikou hardwarové akcelerace filtru je způsob využívající distribuovanou aritmetiku, jenž patří mezi významné metody používané v FPGA. Aplikuje se při součtu součinů

$$y = \langle c, x \rangle = \sum_{n=0}^{N-1} c[n]x[n] \tag{3.1}$$

tedy např. při konvoluci využívané při filtraci obrazu. Důležitým předpokladem je, abychom koeficienty c[n] znali předem, a potom při součinu c[n]x[n] docházelo k násobení konstantou [12].

Koeficienty c[n] jsou tedy známé konstanty a x[n] jsou proměnné, které můžeme zapsat následovně

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \times 2^b \tag{3.2}$$

kde $x_b[n]$ je b-tý bit vzorku x[n]. Výstup y potom můžeme zapsat

$$y = \sum_{n=0}^{N-1} c[n] \times \sum_{b=0}^{B-1} x_b[k] \times 2^b \tag{3.3}$$

a po přeskládání pořadí sčítání dostaneme

$$\begin{aligned}
&y = c[0]\left(x_{B-1}[0]2^{B-1} + x_{B-2}[0]2^{B-2} + \ldots + x_0[0]2^0\right) \\
&+ c[1]\left(x_{B-1}[1]2^{B-1} + x_{B-2}[1]2^{B-2} + \ldots + x_0[1]2^0\right) \\
&\vdots \\
&+ c[N-1]\left(x_{B-1}[N-1]2^{B-1} + \ldots + x_0[N-1]2^0\right) = \\
&= \left(c[0]x_{B-1}[0] + c[1]x_{B-1}[1] + \ldots + c[N-1]x_{B-1}[N-1]\right)2^{B-1} \\
&+ \left(c[0]x_{B-2}[0] + c[1]x_{B-2}[1] + \ldots + c[N-1]x_{B-2}[N-1]\right)2^{B-2} \\
&\vdots \\
&+ \left(c[0]x_0[0] + c[1]x_0[1] + \ldots + c[N-1]x_0[N-1]\right)2^0
\end{aligned} \tag{3.4}$$

nebo po zapsání pomocí sum

$$y = \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{N-1} \underbrace{c[n] \times x_b[n]}_{f(c[n], x_b[n])} = \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{N-1} f\left(c[n], x_b[n]\right) \tag{3.5}$$

Důležitá je správná implementace funkce $f(c[n]x_b[n])$. Nejlepší způsob realizace této funkce je pomocí jedné vyhledávací tabulky (look-up table, LUT), která může být realizovaná jako paměť ROM, v níž jsou uloženy vypočítané částečné výsledky pro jednotlivé bity. Do tabulky vstupují N-bitové vektory $x_b = [x_b[0], x_b[1], \ldots, x_b[N - 1]]$, podle kterých se vyhledá správný výstup. Celou distribuovanou aritmetiku nejlépe pochopíme na jednoduchém příkladu (podle [1]):

Mějme rovnici $y = \sum_{n=0}^{2} c[n]x[n]$, kde c[n] jsou tříbitové koeficienty, které mají následující hodnoty c[0] = 2, c[1] = 3 a c[2] = 1. Vyhledávací tabulka implementující funkci $f(c[n]x_b[n])$ bude vypadat takto:

$x_b[2]$	$x_b[1]$	$x_b[0]$	f(c[n],x[n])
0	0	0	$1 \times 0 + 3 \times 0 + 2 \times 0 = 0_{10} = 000_2$
0	0	1	$1 \times 0 + 3 \times 0 + 2 \times 1 = 2_{10} = 010_2$
0	1	0	$1 \times 0 + 3 \times 1 + 2 \times 0 = 3_{10} = 011_2$
0	1	1	$1 \times 0 + 3 \times 1 + 2 \times 1 = 5_{10} = 101_2$
1	0	0	$1 \times 1 + 3 \times 0 + 2 \times 0 = 1_{10} = 001_2$
1	0	1	$1 \times 1 + 3 \times 0 + 2 \times 1 = 3_{10} = 011_2$
1	1	0	$1 \times 1 + 3 \times 1 + 2 \times 0 = 4_{10} = 100_2$
1	1	1	$1 \times 1 + 3 \times 1 + 2 \times 1 = 6_{10} = 110_2$

Vstupní vzorky x[n] mají hodnoty $x[0] = 1_{10} = 001_2$, $x[1] = 3_{10} = 011_2$ a $x[2] = 7_{10} = 111_2$. Výpočet výstupu y bude následující

Krok t	$x_t[2]$	$x_t[1]$	$x_t[0]$	$f(c[n],x[n]) \times 2^t$ +	**ACC[t - 1]**	**= ACC**
0	1	1	1	6×2^0 +	0	= 6
1	1	1	0	4×2^1 +	6	= 14
2	1	0	0	1×2^2 +	14	= **18**

Pro kontrolu ještě přidám numerické řešení

$$y = c[0] \times x[0] + c[1] \times x[1] + c[2] \times x[2] = 2 \times 1 + 3 \times 3 + 1 \times 7 = \mathbf{18} \quad (3.6)$$

Výsledek jsme vypočítali pouze za použití operací sčítání, bitový posun a vyhledání v LUT tabulce. V hardwaru se ještě často bitový posun výsledku z LUT tabulky o b bitů nahrazuje posunem celého obsahu akumulátoru o 1 bit v každém kroku. Ušetří se tím použití drahého válcového posouvače (barrel shifter)[1].

3.3.1 Znaménková distribuovaná aritmetika

U čísel se znaménkem je distribuovaná aritmetiky trochu jiná. Musíme počítat s tím, že nejvyšší bit je znaménkový. Proměnné x[n], které jsou reprezentovány B + 1 bity, kde nejvyšší bit je znaménkový, můžeme zapsat

$$x[n] = -2^b \times x_B[n] + \sum_{b=0}^{B=1} x_b[n] \times 2^b \quad (3.7)$$

Po stejných úpravách jako u bezznaménkové distribuované aritmetiky dostaneme výstup

$$y = -2^b \times f(c[n], x_B[n]) + \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{N-1} f(c[n], x_b[n]) \quad (3.8)$$

Pro snadnější pochopení opět uvedu konkrétní příklad (podle [1]). Mějme rovnici $y = \sum_{n=0}^{2} c[n]x[n]$, kde c[n] jsou konstantní koeficienty, které mají následující hodnoty c[0] = -2, c[1] = 3 a c[2] = 1. Vyhledávací tabulka implementující funkci $f(c[n]x_b[n])$ bude vypadat takto:

$x_b[2]$	$x_b[1]$	$x_b[0]$	**f(c[n],x[n])**
0	0	0	$1 \times 0 + 3 \times 0 - 2 \times 0 = 0_{10}$
0	0	1	$1 \times 0 + 3 \times 0 - 2 \times 1 = -2_{10}$
0	1	0	$1 \times 0 + 3 \times 1 - 2 \times 0 = 3_{10}$
0	1	1	$1 \times 0 + 3 \times 1 - 2 \times 1 = 1_{10}$
1	0	0	$1 \times 1 + 3 \times 0 - 2 \times 0 = 1_{10}$
1	0	1	$1 \times 1 + 3 \times 0 - 2 \times 1 = -1_{10}$
1	1	0	$1 \times 1 + 3 \times 1 - 2 \times 0 = 4_{10}$
1	1	1	$1 \times 1 + 3 \times 1 - 2 \times 1 = 2_{10}$

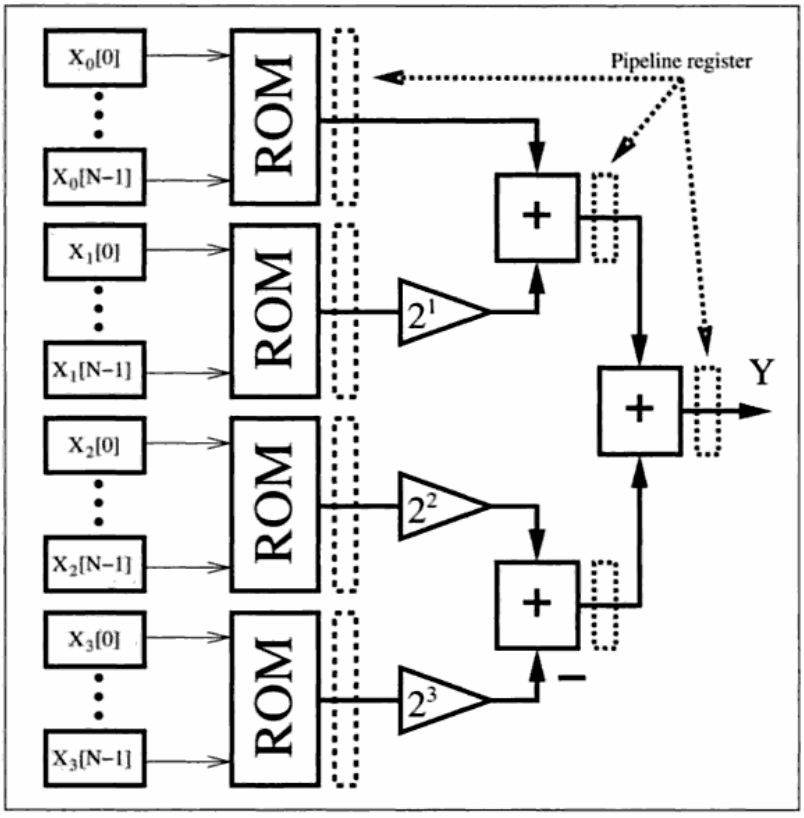
Vstupní 4-bitové vzorky x[n] mají hodnoty x[0] = 1_{10} = 0001_2, x[1] = -3_{10} = 1101_2 a x[2] = 7_{10} = 0111_2. Výpočet výstupu y bude následující

Krok t	**x_t[2]**	**x_t[1]**	**x_t[0]**	**$f(c[n],x[n]) \times 2^t$ + Y[t - 1] = Y**
0	1	1	1	2×2^0 + 0 = 2
1	1	0	0	1×2^1 + 2 = 4
2	1	1	0	4×2^2 + 4 = 20
				$-f(c[n],x[n]) \times 2^t$ + Y[t - 1] = Y
3	0	1	0	-3×2^3 + 20 = **-4**

Pro kontrolu ještě přidám opět numerické řešení

$$y = c[0] \times x[0] + c[1] \times x[1] + c[2] \times x[2] = -2 \times 1 + 3 \times (-3) + 1 \times 7 = \mathbf{-4} \qquad (3.9)$$

Při implementaci znaménkové distribuované aritmetiky musíme správně rozhodnout, kdy se bude přičítat a kdy odčítat. První možnosti je akumulátor, který si bude kontrolovat, jestli se má přičíst nebo odečíst, druhou možností je LUT tabulka s jedním vstupem navíc. Tato možnost ale není výhodná, protože se zdvojnásobí počet uložených hodnot v tabulce [1].



Obrázek 3.4: Paralelní implementace distribuované aritmetiky [1]

3.3.2 Modifikace distribuované aritmetiky

Distribuovanou aritmetiku můžeme dále modifikovat dvěmi způsoby. První možností je zmenšení velikosti celého návrhu a druhou zvýšení rychlosti výpočtu.

Čím více koeficientů bude mít systém, tím větší bude potřeba LUT tabulka, protože bitová šířka vstupu se rovná počtu koeficientů. Velikost LUT tabulky roste dokonce exponenciálně. Celý systém

zmenšíme, pokud místo jedné velké tabulky použijeme více menších. Výstupy těchto tabulek se pak pouze sečtou. Jestliže přidáme ještě pipeline registry, tak se příliš nesníží ani rychlost [1].
Předpokládejme počet koeficientů LN, pak při použití jedné tabulky platí

$$y = \langle c, x \rangle = \sum_{n=0}^{LN-1} c[n]x[n] \tag{3.10}$$

Pokud tabulku nahradíme L menšími tabulkami s N bitovým vstupem, pak pro výstup platí

$$y = \langle c, x \rangle = \sum_{l=0}^{L-1} \sum_{n=0}^{N-1} c[Ll + n]x[Ll + n] \tag{3.11}$$

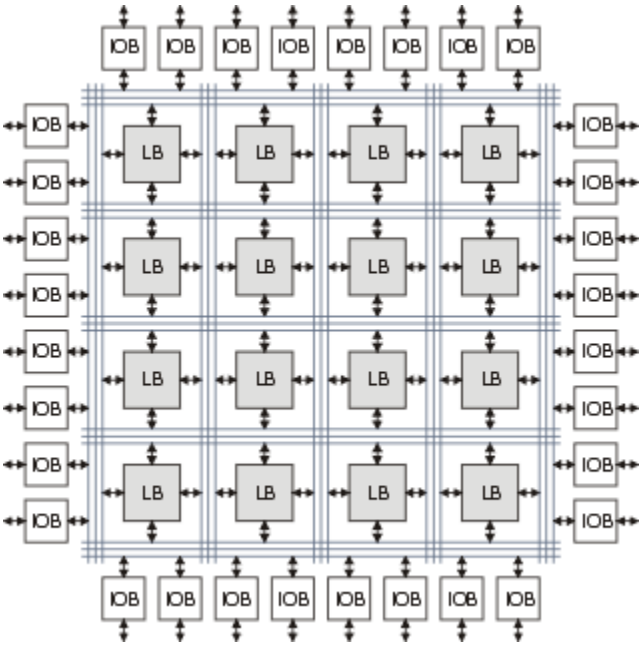
Z předcházejících vztahů vidíme, že velikost tabulky se zmenšila z jedné $2^{LN} \times$ B LUT na čtyři $2^{N} \times$ B tabulky [1].

Další modifikace rozložené aritmetiky je za účelem zvýšení rychlosti. Základní metoda přijímá jeden bit od každé vstupní hodnoty x[n]. Pokud bude přijímat dva bity najednou, rychlost výpočtu se zdvojnásobí. Nejvyšší rychlost dosáhneme, jestliže použijeme paralelní zřetězené zpracování, kdy bude systém přijímat všechny bity vstupních dat najednou. Pro maximální rychlost použijeme pro každý bitový vektor samostatnou LUT tabulku. Všechny tabulky budou mít ale stejný obsah. Při zvětšení bitové šířky vstupu musíme zvýšit i počet LUT tabulek. Jestliže nepotřebujeme příliš mnoho koeficientů a vstup má přijatelnou bitovou šířku je tato modifikace velmi výkonná [1].

4 Hardwarové prostředky

4.1 FPGA

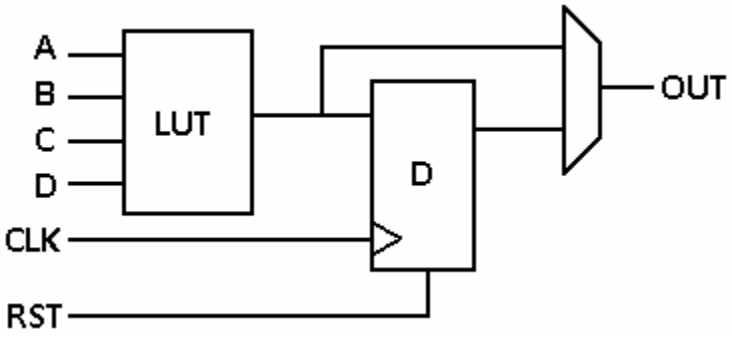
FPGA (Field Programmable Gate Array), nebo-li programovatelné hradlové pole se skládá z programovatelných bloků, které se označují CLB (Configurable logic block) někdy jen LB (logic block). Tyto bloky můžou být navzájem různě propojeny globální propojovací maticí. Každý blok se skládá z menších částí označovaných anglickým slovem slice. Slice se skládá z Look Up tabulky (LUT), klopného obvodu typu D a multiplexoru [5]. Kromě CLB obsahuje FPGA ještě vstup-výstupní obvody IOB, které zajišťují připojení k externím pinům pouzdra.



Obrázek 4.1: Vnitřní struktura FPGA [13]

Některé sousední CLB bloky lze propojit bez nutnosti použití globální propojovací matice. Tyto signály pak mají mnohem menší zpožděni a využívají se na realizaci např. rychlých obvodů šíření přenosu [13]. Kromě propojovacích signálů se na čipu nalézá ještě sběrnice rozvádějící hodinový signál. Je realizována tak, aby měly její vodiče co nejmenší zpoždění.

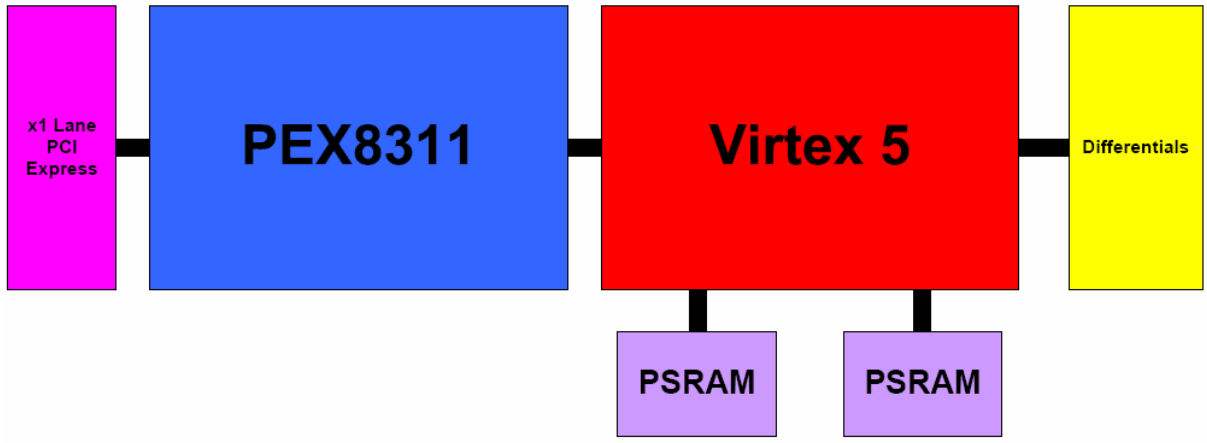
Mimo zmíněné bloky obsahuje FPGA často ještě další speciální integrované komponenty, protože jejich realizace pomocí CLB by nebyla efektivní a zabrala by velkou plochu obvodu. Nejčastěji se jedná o zabudované blokové paměti, násobičky, PLL (Phase Locked Loop) obvody a podobně.



Obrázek 4.2: Jedna slice uvnitř CLB

4.2 Použitá platforma

Pro praktickou realizaci obrazového filtru byla vybrána platforma od firmy Pico Computing, která se zabývá integrací FPGA obvodů do karet typu CompactFlash, CardBus a ExpressCard. Konkrétně byla vybrána karta Pico E-16 typu ExpressCard 34 mm, kvůli dostupným hardwarovým prostředkům. Karta je vhodná pro náročné operace, mezi než např. patří digitální zpracování signálů, zpracování videa, komprese, šifrování, dešifrování a mnoho dalších vědeckých výpočtů.



Obrázek 4.3: Schéma karty Pico E-16 [6]

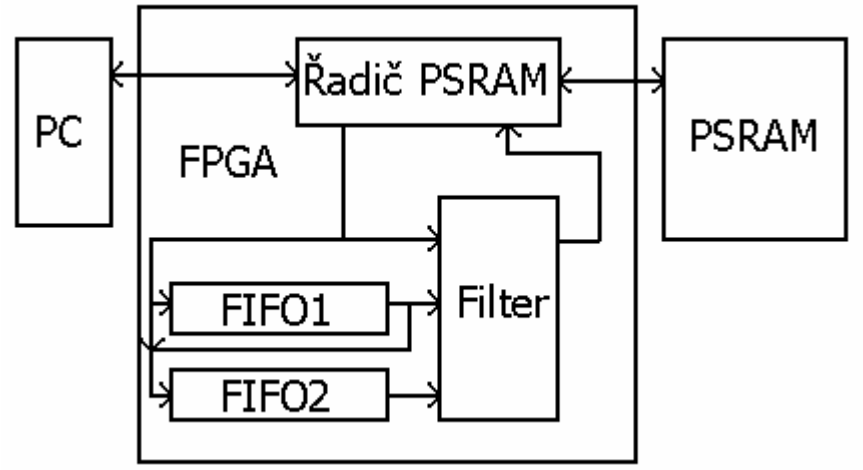
Hlavní součástí je výkonné hradlové pole FPGA s označením Virex-5 LX50 od firmy Xilinx, které obsahuje vysokorychlostní DSP slices, které se využívají při zpracování signálů. Virtex-5 patří mezi výkonější obvody a obsahuje 28 800 programovatelných bloků CLB, 1 728 kb blokové paměti, 46 080 DSP slice a jeho Look Up tabulky obsahují šest vstupů [6].

Další součástí vybrané platformy jsou dvě paměti typu PSRAM o celkové velikosti 32 MB (2×16 MB). PSRAM (Pseudostatic RAM) je dynamická paměť obsahující vestavěný obvod na refresh. Kombinuje velkou hustotu záznamu dynamické paměti s jednoduchostí statické paměti [14]. PSRAM na Pico kartě umožňuje práci ve 3 režimech, v asynchronním, stránkovém a synchronním burst režimu.

Karta je připojena přes sběrnici PCI Express o rychlosti $\times 1$. Komunikaci mezi sběrnicí PCI Express a vnitřní sběrnicí zajišťuje bridge PEX8311. Vnitřní sběrnice pracuje na frekvenci 66 MHz a je paralelní. Ke kartě lze připojit další periferní zařízení přes diferenciální rozhraní. Rozhraní obsahuje 9 párů diferenciálních signálů, které mohou být použity i jako 18 samostatných nezávislých signálů [6].

5 Praktická realizace

V jazyce VHDL byl vytvořen kompletní obrazový filtr o velikosti 3×3 určený pro platformu popsanou v předchozí kapitole. K tomu byla vytvořena softwarová aplikace, která se stará o načítání obrazu ze souboru a o zpětné uložení filtrovaného obrazu. Celá architektura je schopná zpracovat libovolný obraz v odstínech šedi uložený v mnoho různých formátech o maximální šířce 4094 pixelů nebo o maximální velikosti 32 megapixelů. Tato omezení jsou dána maximální kapacitou fronty a maximální kapacitou hardwarové paměti umístěné na zvolené platformě. Zjednodušené schéma celé architektury zobrazuje obrázek.

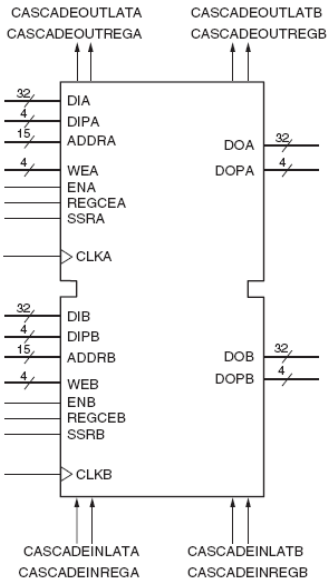


Obrázek 5.1: Schéma navržené architektury

Po startu softwarové aplikace se inicializují koeficienty filtru, které se následně zapíší do registru uvnitř FPGA. Dále se načte filtrovaný obraz a uloží se do paměti připojené k FPGA. Jakmile je celý obraz uložen v paměti spustí se filtrace. Postupně se načítají jednotlivé body z paměti, posílají se do filtru a zpět se ukládají do paměti na původní místo. Fronty FIFO se používají na uložení dat s obrazovými body předchozích dvou řádků. V následující části práce se zaměřím na podrobnější popis jednotlivých komponent celé architektury.

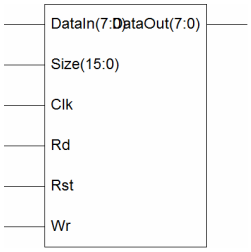
5.1 Fronta FIFO

Frontu je v navržené implementaci zapotřebí implementovat ve dvou instancích. Obě fronty ukládají předchozí řádky obrazu. Nejlepší možností pro implementaci se jeví použití blokové paměti RAMB18 nebo RAMB36. Vybrána byla větší paměť RAMB36, aby byla celá architektura schopná zpracovat obraz o větší šířce. Maximální kapacita paměti činí 36 Kb a může mít tyto datové šířky 1, 2, 4, 9, 18 a 36 bitů. Protože filtr pracuje s daty o velikosti 8 bitů, byla vybrána datová šířka 9 bitů, přičemž devátý paritní bit není využit. Fronta může tedy uložit maximálně 4096 osmibitových hodnot, takže její kapacita je 4096 bytů.



Obrázek 5.2: Komponenta RAMB36

Jedná se o dvouportovou paměť (viz obrázek 5.2), což má výhodu, že můžeme současně číst i zapisovat. Problém může nastat, pokud bychom chtěli číst i zapisovat na stejnou adresu. V navržené architektuře k tomu ale nikdy nedojde, protože první se vždy přečte hodnota z určité adresy a v příštím taktu se přepíše novou hodnotou. Porty A jsou implementovány jako porty zápisové a porty B jako čtecí.



Obrázek 5.3: Rozhraní navržené fronty

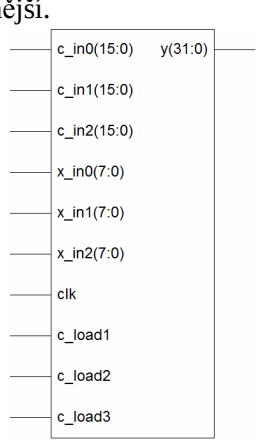
Na obrázku 5.3 je zobrazeno rozhraní implementované fronty. Vstupní data se přivádí na sběrnici DataIn a výstupní data jsou připojena na sběrnici DataOut. Data se zapisují do fronty s náběžnou hranou, pokud je signál Wr nastaven na hodnotu 1. Čtení probíhá pomocí signálu Rd. Je-li jeho hodnota nastavena na 1, s náběžnou hranou se přečtou data z fronty. Platná hodnota se na výstupu objeví až o takt pozdíš, a proto můžeme data ze sběrnice DataOut přečíst až s příští náběžnou hranou. O správné adresy, kam se má zapisovat nebo kde se má číst, se starají dva ukazatele (čítače), které se inkrementují po každém zápisu/čtení. Na sběrnici Size je přivedena velikost aktuální fronty. Pokud ukazatele adres dosáhnou hodnoty Size jsou nastaveny jejich adresy na nulu. Fronta si nijak nehlídá svoji velikost. Pokud budeme ukládat do fronty více dat, než je její velikost, data se začnou od začátku přepisovat. Také velikost přivedená na sběrnici Size není v bloku fronty kontrolována. Ke kontrole dochází pouze v softwaru, který se stará o načítaní filtrovaného obrazu.

Po nastavení signálu Rst na hodnotu 1 dojde k vynulování celého obsahu fronty (paměti) a k nastavení ukazatelů adres na nulovou adresu.

5.2 Realizace 2D filtru

Hlavní částí celé architektury je 2D FIR filtr s filtrovacím jádrem o rozměrech 3×3 obrazových bodů. Filtr využívá metodu distribuované aritmetiky (distributed arithmetic) vylepšenou o paralelní zřetězené zpracování. Protože vstupní vzorky náleží do intervalu od 0 do 255, byla použita bezznaménková distribuovaná aritmetika. Výsledek pro každý řádek se spočítá podle algoritmu uvedeného v kapitole 3.3 viz obrázek 3.4, a pak se tyto tři částečné výsledky sečtou.

Metoda distribuované aritmetiky byla vybrána proto, aby bylo možné snadno měnit koeficienty filtru. Pokud LUT tabulku naimplementujeme pomocí registrů a jednoho multiplexoru, můžeme pak snadno měnit koeficienty, tedy obsah tabulky (obsah jednotlivých registrů). Při použití RAG algoritmu by bylo řešení o hodně obtížnější.



Obrázek 5.4: Rozhraní navrženého filtru

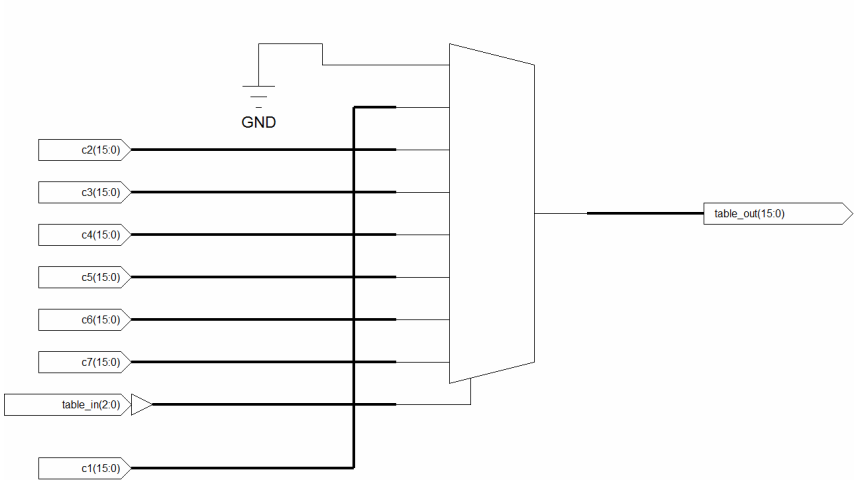
Na začátku, než filtr začne přijímat vzorky dat, se musí načíst koeficienty filtru. K tomu slouží sběrnice c_in0, c_in1 a c_in2, pomocí kterých se načítají koeficienty postupně pro jednotlivé řádky filtru. Jestliže má signál c_load1 hodnotu 1, načtou se s náběžnou hranou koeficienty pro první řádek filtru přes sběrnice c_in a zároveň se spočtou hodnoty pro Look Up tabulky, které potřebuje filtr pro první řádek. Obdobná situace nastává i pro signály c_load2 a c_load3, kdy se načítají koeficienty a počítají se hodnoty pro Look-Up tabulky pro druhý a třetí řádek filtru.

Sběrnice c_in mají datovou šířku 16 bitů a obsahují hodnoty zapsané pomocí dvojkového doplňku, tedy nejvyšší bit je znaménkový. Přitom spodních 8 bitů reprezentuje desetinou část a horních 8 bitů celou část koeficientu. Koeficienty můžou být tedy v rozptylu od –128 do 127 s přesností 1/256.

Pokud není ani jeden ze signálů c_load na hodnotě 1, tak filtr přijímá s náběžnou hranou data ze sběrnic x_in0, x_in1, x_in2 a počítá výstupní hodnotu. Každá sběrnice x_in patří pouze k jednomu řádku filtru a každý řádek provádí výpočty samostatně. Od všech vzorků se vezmou zvlášť jednotlivé bity, tedy nulté, první až sedmé, a vyhledají se k nim odpovídající hodnoty v Look Up tabulkách. Všemi vzorky jsou myšleny předchozí vstupní hodnoty a aktuální hodnota na sběrnici x_in. Pokud se načetl zatím pouze jeden vzorek pro každý řádek, jsou předchozí vstupní hodnoty nulové. V dalším kroku dojde k logickému bitovému posunu výstupních hodnot z Look Up tabulek. Hodnoty se posunou doleva o tolik bitů, kolikáté bity vstupovaly do Look-Up tabulky, např. pokud vstupem do LUT byly páté bity, posune se výstupní hodnota z LUT o 5 bitů doleva, nebo-li bude vynásobena hodnotou 2^5. Pak se tyto posunuté hodnoty sečtou a dostaneme výsledky pro jednotlivé řádky. Posledním krokem je součet výsledků všech řádků. Všechny kroky se realizují s náběžnou hranou hodinového signálu. Výstupní data mají šířku 32 bitů, z čehož spodních 8 bitů reprezentuje desetinou část, která bude později stejně zahozena.

5.2.1 Look Up tabulka

Protože jsem pro implementaci zvolil techniku rozložené aritmetiky, je potřeba realizovat Look Up tabulku. Každý bit vstupních dat vyžaduje jednu LUT. Vstupní data pro jednotlivé řádky mají šířku 8 bitů, a protože filtr obsahuje 3 řádky, tak celkem potřebuje architektura 24 Look Up tabulek. Přitom všechny tabulky pro jeden řádek obsahují stejná data.



Obrázek 5.5: Vnitřní struktura Look Up tabulky

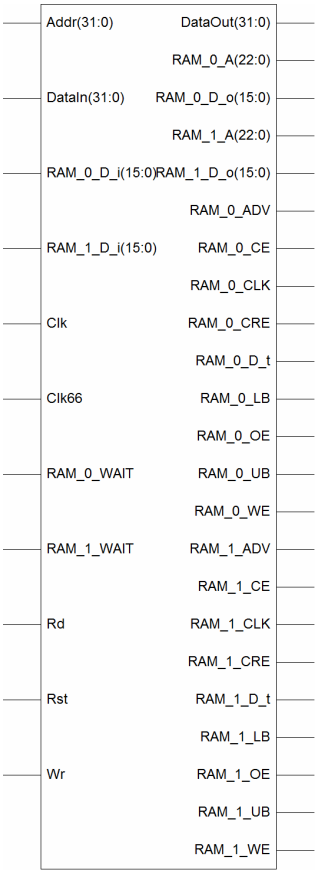
Na rozdíl od návrhu na obrázku 3.4 není tabulka realizována jako paměť ROM, ale jako jeden multiplexor, aby bylo možné měnit snadno koeficienty. Na vstupní sběrnice c (viz Obrázek 5.5) jsou přivedeny hodnoty vypočtené z koeficientů ve filtru. Podle vstupu table_in se pak přepínají tyto hodnoty na výstupní sběrnici table_out.

5.3 Řadič paměti

Mezi složitější části architektury patří řadič paměti PSRAM. Paměť disponuje třemi režimy práce, jenž jsou synchronní burst, asynchronní a stránkový. V navržené architektuře se využívá pouze režim synchronní. Na počátku je ale paměť v asynchronním režimu a musí se nastavit na synchronní.

Implementace řadiče vychází a rozšiřuje implementaci řadiče dodávaného k Pico kartě. Tento řadič obsahuje pouze ovládání asynchronního režimu a je přístupný pouze z počítače přes Pico sběrnici. Aby se dalo číst a zapisovat i zevnitř z FPGA, byla tato část přesunuta do uživatelského modulu (do mé navržené architektury) a tím upraveny i signály v dodávané implementaci. Navíc jsem tento řadič rozšířil o synchronní burst režim.

Rozhraní řadiče paměti je znázorněno na obrázku 5.6. Signály a sběrnice RAM jsou napojeny na hardwarovou paměť, ostatní směřují do navržené architektury. Některé signály RAM prošly změnou, tak aby všechny byly aktivní na hodnotě 1.



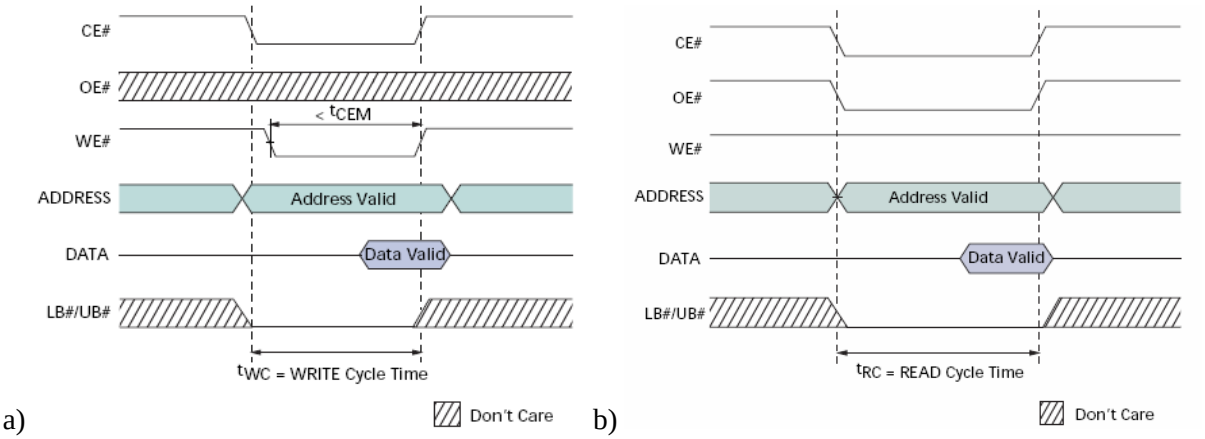
Obrázek 5.6: Rozhraní řadiče paměti

5.3.1 Asynchronní režim

Jak bylo napsáno v úvodu kapitoly 5, asynchronní režim se používá v navržené architektuře pouze na přepnutí paměti do synchronního burst režimu. Kromě toho byl tento režim využit i při ladění synchronního režimu.

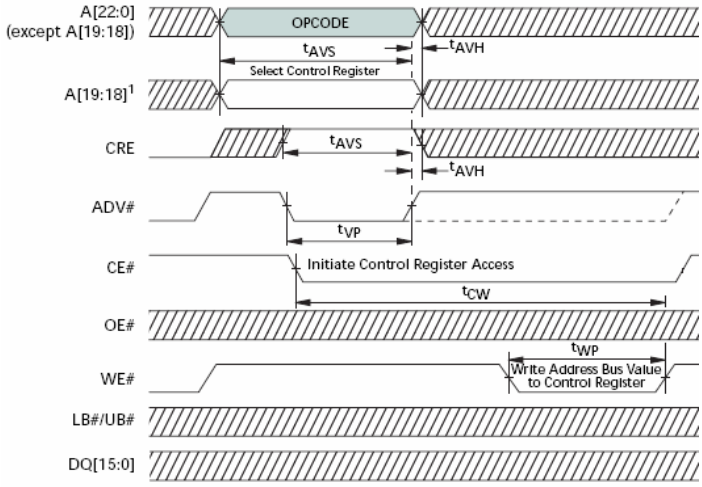
Před vlastním zápisem do paměti potřebuje řadič znát adresu zápisu a data, která se budou zapisovat. Zápis adresy do řadiče se uskuteční s náběžnou hranou hodinového signálu přes sběrnici DataIn, je-li signál Wr na hodnotě 1, a pokud adresa na sběrnici Addr se rovná hexadecimální hodnotě 0x5100000C. Zápis dat do řadiče se provádí stejným způsobem, pouze adresa na sběrnici Addr se rovná hodnotě 0x51000008. Teprve potom se může provést samotný zápis z řadiče do paměti. Zápis se spustí zapsáním hodnoty 0x00000001 na adresu 0x51000000. Řadič aktivuje potřebné signály, vyčká určitou dobu, než se data zapíší, a deaktivuje signály.

Obdobným způsobem probíhá i čtení z paměti. Nejprve se na adresu 0x5100000C zapíše adresa odkud se bude číst, pak se spustí samotné čtení z paměti zápisem hodnoty 0x0000000 na adresu 0x5100000 a nakonec se z adresy 0x51000004 na výstupní sběrnici DataOut přečtou správná data. Samotné čtení probíhá úplně stejně jako zápis, pouze se aktivují jiné signály.



Obrázek 5.7: Časový diagram asynchronního a) zápisu, b) čtení [7]

K nastavení synchronního burst režimu slouží registr BCR (Bus Configuration Register). Zápis do konfiguračních registrů probíhá podobně jako zápis do paměti. Nepotřebujeme ale žádná data, protože do registru se přenese hodnota zadané adresy. Abychom poznali, kdy se má uskutečnit zápis do konfiguračních registrů, obsahuje paměť signál CRE. V řadiči se aktivuje tento signál, pokud je na adresu 0x51000000 zapsána hodnota 0x00000003 místo hodnoty 0x00000001 (rozhodující je první bit). Do kterého registru má paměť zapsat, pozná podle 18. a 19. bitu adresy. Zápis do registru BCR lze vidět na obrázku 5.8. Podrobnější informace lze nalézt v [7].



Obrázek 5.8: Časový diagram zápisu do konfiguračního registru [7]

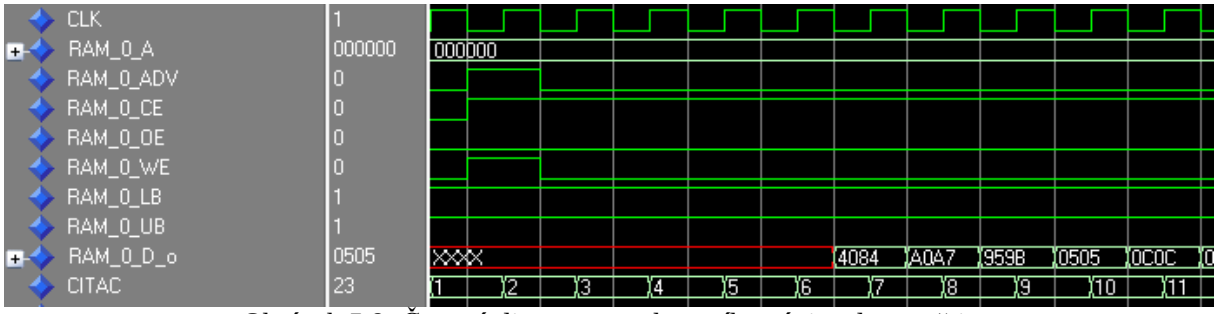
5.3.2 Synchronní burst režim

Řadič pro synchronní burst režim není kompletně vyřešen, neboť jeho úplná realizace by rozsahem odpovídala celé bakalářské práci. Implementovány jsou pouze části potřebné v navržené architektuře. Nelze tedy využít všechny možnosti a nastavení paměti.

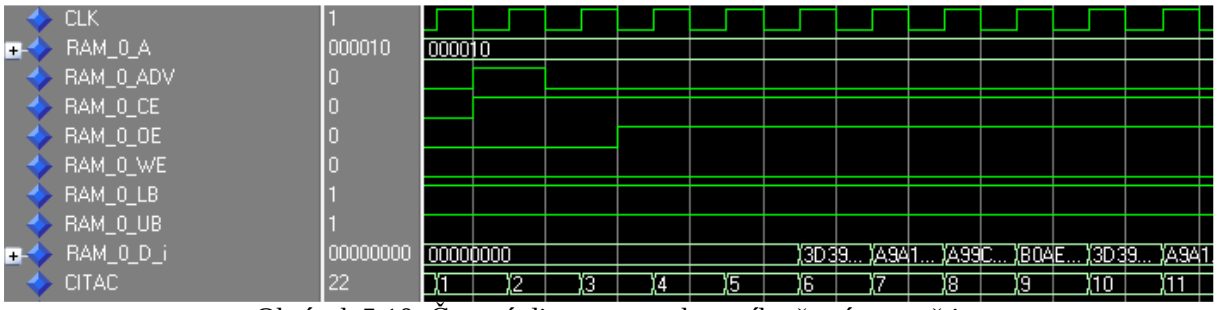
Paměť pracuje s daty o velikosti 32 bitů a může číst nebo zapisovat po 4, 8, 16, 32 a nebo až 128 slovech. V navržené implementaci se využívá přenos dat po 16 slovech. Před přenosem dat se musí nastavit počáteční adresa, od které se bude číst nebo zapisovat. Provádí se to stejně jako v asynchronním režimu, zapsáním na adresu 0x5100000C. Poté se může spustit operace zápisu nebo čtení, která se aktivuje zápisem hodnoty 0x00000001 nebo 0x00000000 na adresu řadiče 0x51000000. V řadiči se spustí čítač, podle jehož hodnot se nastavují signály směřující do paměti (viz obrázky 5.9 a 5.10). Při zápisu do paměti očekává řadič s každou náběžnou hranou data na adrese

0x51000008 a při čtení se na adrese řadiče 0x51000004 opět s náběžnou hranou objevují výstupní data.

Před začátkem přenosu dat má však paměť určité zpoždění, jehož velikost lze ovlivnit obsahem registru BCR. Velikost zpoždění může být, buď proměnná nebo konstantní po celou dobu práce paměti. V navržené architektuře je zvoleno konstantní zpoždění a jeho délka činí 5 taktů, což je nejmenší možná hodnota pro zvolený hardware (nebo-li pro jeho frekvenci 66 MHz). Při konstantním zpoždění se nemusí kontrolovat signály WAIT (RAM_0_WAIT a RAM_1_WAIT) a nemělo by dojít ke kolizím při obnovování (refresh) paměti. Více informací o hardwarové paměti naleznete v [7].



Obrázek 5.9: Časový diagram synchronního zápisu do paměti



Obrázek 5.10: Časový diagram synchronního čtení z paměti

5.4 Toplevel architektura

Modul toplevel obsahuje všechny části zmíněné v předchozích kapitolách a řídí chod celého filtru. Hlavní částí je jeden velký konečný automat, který pouze sekvenčně prochází jednotlivé stavy. Přechod se vždy uskuteční, jakmile se dokončí operace prováděná v daném stavu.

Po zapsání konfigurační informace do FPGA a jeho nakonfigurování, tedy po spuštění celé aplikace, dojde k počáteční inicializaci. Hlavní operací inicializace je zápis do konfiguračního registru paměti BCR (Bus Configuration Register). Zde se nastaví synchronní burst režim paměti, délka zpoždění na konstantní hodnotu 5 taktů a počet přenášených dat na 16 slov. Po inicializaci systém čeká dokud se neozve softwarová aplikace.

Architektura nejprve očekává 32-bitovou hodnotu obsahující velikosti filtrovaného obrazu. Rozměry si uloží do registrů a čeká na příjem koeficientů filtru. V každé 32-bitové hodnotě jsou uloženy dva koeficienty, pouze v poslední se nachází na spodních 16 bitech jeden devátý koeficient. Podrobný přehled lze vidět v tabulce 5.1.

Operace	Adresa	Data	
		Bity 31 až 16	Bity 15 až 0
Zápis rozměrů obrazu	0x10000000	Šířka obrazu	Výška obrazu
Zápis koeficientů (1. a 2.)	0x11000000	1. koeficient	2. koeficient
Zápis koeficientů (3. a 4.)	0x11000004	3. koeficient	4. koeficient
Zápis koeficientů (5. a 6.)	0x11000008	5. koeficient	6. koeficient
Zápis koeficientů (7. a 8.)	0x1100000C	7. koeficient	8. koeficient
Zápis posledního koeficientu (9.)	0x11000010	nevyužito	9. koeficient

Tabulka 5.1: Komunikace mezi softwarem a FPGA při přenosu rozměrů obrazu a koeficientů

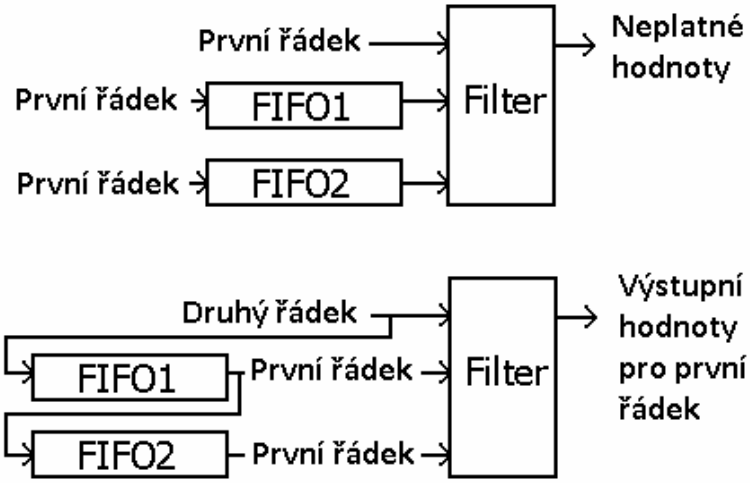
Jakmile přijme poslední koeficient, přejde do následujícího stavu, kde se ukládá filtrovaný obraz do paměti. Přenos probíhá zápisem dat na adresu 0x2XXXXXXX, kde XXXXXXX u prvního přenášeného slovo určuje počáteční adresu v paměti, kam se data zapíší (podrobnosti v tabulce 5.2). Aby bylo zajištěno správné načasování vstupních dat do paměti, uloží se nejprve data do pomocného bufferu, z kterého jsou následně uložena do paměti. Přenos z bufferu do paměti se odstartuje zapsáním hodnoty 0x00000001 na adresu 0x51000000. Způsob zápisu do paměti je popsán v kapitole 5.3.2. V jednom zápisovém cyklu se uloží 16 32-bitových hodnot. Pokud se ze softwarové aplikace přenese více než 16 hodnot, začnou přebytečné hodnoty od začátku přepisovat pomocný buffer. Obsah prvního bufferu se navíc zkopíruje do druhého bufferu, aby se při začátku filtrace nemusely tyto data načítat z paměti. Po každém uložení dat do paměti se kontroluje, kolik již bylo uloženo. Pokud se již celý obraz nachází v paměti, postoupí se do dalšího stavu, kde se vloží načtené koeficienty do filtru. V třech taktech se uloží koeficienty postupně pro všechny tři řádky filtru. Pak se přejde do následujícího stavu, kde se již konečně spustí filtrace obrazu.

Aby výstupní obraz měl správné hodnoty bodů po obvodu, musí se krajní body replikovat. Proto při vkládání hodnot prvního řádku obrazu se tyto data vkládají také do obou front (viz obrázek 5.11). První správné hodnoty se potom na výstupu filtru objeví se zpožděním t_Z, které je dáno následujícím výrazem

$$t_Z = t_R + t_F \tag{5.1}$$

kde t_R je doba filtrace jednoho řádku a t_F je zpoždění filtru.

K replikaci dochází také vždy na konci a začátku řádku, kdy se pouze poslední nebo první bod řádku vloží do filtru dvakrát. Kvůli tomu lze filtrovat obraz o maximální velikosti 4094 a ne 4096, protože fronty obsahují krajní body dvakrát.



Obrázek 5.11: Vkládání dat do/z front na počátku filtrace, nahoře – vkládání prvního řádku, dole vkládání druhého řádku

Zatímco se do filtru posílají data z jednoho bufferu, do druhého bufferu se z paměti načítají další data, aby byla ihned k dispozici a filtr nemusel čekat (filtr to ani neumí). Obdobným způsobem se realizuje i zápis výstupních hodnot do paměti. Data se opět ukládají do bufferu, a jakmile je plný, jeho obsah se zkopíruje do druhého bufferu, který se uloží do paměti, než se znovu zaplní první buffer.

Po vložení všech hodnot do filtru, se přejde do následujícího stavu a na vstup se přivádí hodnoty posledního řádku (obsah první fronty), kvůli replikaci krajních bodů. V tomto stavu se pouze ukládají výstupní hodnoty filtru pro poslední řádek. V dalších stavech se zápis dokončí, vynulují se fronty a nastaví se v registru testovatelným ze softwarové aplikace, že proces filtrace byl dokončen. Poté se přejde do počátečního stavu, v kterém můžeme načíst obsah paměti.

Načítání filtrovaného obsahu probíhá obdobným způsobem jako zápis. Nejprve se na adresu 0x2XXXXXXX zapíše adresa, od které se bude číst obsah paměti. Stejně jako při zápisu XXXXXXX určuje tuto adresu. Poté se zápisem hodnoty 0x00000000 na adresu 0x51000000 odstartuje čtení z paměti do pomocného bufferu. Nakonec se z adresy 0x2XXXXXXX přečte obsah bufferu. Přehled celého čtení lze vidět v tabulce 5.2.

Krok	**Adresa Čtení/Zápis**	**Data**	**Popis**
Zápis obrazu ze softwarové aplikace do paměti na Pico kartě			
1	0x2XXXXXXX Zápis	32-bitová hodnota obsahující 4 body obrazu	Zápis dat ze softwaru do pomocného bufferu XXXXXXX – počáteční adresa v paměti, od které se začne zapisovat (pouze u 1. zapsané hodnoty) Tento krok se opakuje 16krát, dokud se nezaplní buffer.
2	0x51000000 Zápis	0x00000001	Spuštění zápisu z bufferu do paměti
Čtení obrazu z paměti na Pico kartě do softwarové aplikace			
1	0x2XXXXXXX Zápis	libovolná	Zápis adresy XXXXXXX – počáteční adresa v paměti, od které se začne číst
2	0x51000000 Zápis	0x00000000	Spuštění čtení z paměti do bufferu.
3	0x2XXXXXXX Čtení	32-bitová hodnota obsahující 4 body obrazu	Čtení dat z bufferu do softwaru Tento krok se opakuje 16krát, dokud se nepřečte celý buffer.

Tabulka 5.2: Komunikace mezi softwarem a architekturou v FPGA při čtení a zápisu obrazových dat

5.5 Softwarová aplikace

Softwarová aplikace zajišťuje komunikaci s FPGA kartou. Jedná se o jednoduchou konzolovou aplikaci implementovanou v jazyce C++. Mezi hlavní úkoly patří načítání koeficientů filtru, načítání obrazu ze souboru a zpětné uložení filtrovaného obrazu do souboru. K načítání a ukládání obrazu používá aplikace funkce z knihovny OnenCV. Tato knihovna byla použita z důvodu možnosti načtení téměř libovolného obrazového formátu. Po načtení je obraz ještě převeden do odstínů šedi, kdyby náhodou nebyl, protože aplikace umí filtrovat pouze obrazy v odstínech šedi.

Koeficienty můžou být zadány jako reálná čísla v rozsahu od –128 do 127. Po načtení se musí převést na 16-ti bitové celé číslo zapsané ve dvojkovém doplňku. Nejprve se vynásobí hodnotou 256

a odtrhne se desetinná část, čímž dostaneme celé číslo. Abychom ze záporného čísla dostali kladné ve dvojkovém doplňku, tak musíme ještě k číslu přičíst hodnotu 65536. Koeficienty se posílají po dvou zapsány v jedné 32-bitové proměnné, a tak se první koeficient vynásobí číslem 65536 a přičte se k němu druhý koeficient.

Oba rozměry obrazu se také posílají v jedné 32-bitové hodnotě, a tak se musí upravit stejně jako koeficienty (šířka $\times$ 65536 + výška). Po zaslání těchto údajů do FPGA se začne načítat obraz po jednotlivých pixelech. Hodnoty pixelů obrazu jsou v rozmezí od 0 do 255, a tak jsou uloženy po čtyřech v jedné 32-bitové proměnné. Do této proměnné se vkládají stejným způsobem jako koeficienty pouze se předchozí uložená hodnota vynásobí číslem 256. Vždy po načtení 64 pixelů (16×4) se uloží do paměti na Pico kartě. Po načtení a uložení celého obrazu do paměti se automaticky spustí filtrace. Aplikace pak testuje hodnotu na adrese 0x30000000, a pokud je rovna číslu 0x11111111, tak filtrace skončila, což znamená, že může načíst filtrovaný obraz z paměti na Pico kartě.

Načítání dat z paměti probíhá ve dvou krocích. Nejprve se načítají data a ukládají se do pomocného bufferu, a pak se teprve zapisují do souboru. Při načtení obdržíme 32-bitové hodnoty, z kterých potřebujeme dostat jednotlivé osmibitové body obrazu. Provádí se to pomocí zbytku po celočíselném dělení. Z obdržené hodnoty spočítáme zbytek po dělení hodnotou 256, čímž dostaneme jednu osmibitovou hodnotu (jeden bod obrazu). Pak 32-bitobé číslo celočíselně podělíme 256 a opakujeme celou operaci, dokud nezískáme všechny čtyři body obrazu. Obdržené 4 body obrazu jsme ale získali v obraceném pořadí, na což nesmíme zapomenout při zápisu do pomocného bufferu. Po získání všech bodů obrazu se obsah bufferu zapíše do souboru.

6 Vyhodnocení navrženého řešení

Navržený systém byl nejdříve kompletně odsimulován v programu ModelSim, poté vysyntetizován, nahrán do FPGA a experimentálně ověřen. Testování probíhalo na obrazech různých velikostí, od jednotek pixelů až po tisíce pixelů. Bylo použito několik různých filtrovacích masek s kladnými, zápornými i desetinnými koeficienty.

Abych mohl porovnat rychlost realizované architektury, vytvořil jsem ještě jednu čistě softwarovou testovací aplikaci v jazyce C++. Testovací aplikace se chová navenek úplně stejně jako hardwarová aplikace. Implementaci jsem mohl provést dvěmi způsoby. První méně efektivní možností je realizovat celý algoritmus čistě v jazyce C++ za použití standardních knihoven. Já jsem si ale vybral druhou možnost a to použití speciální knihovny, kde je již celý filtr naimplementován, aby byla zajištěna efektivita výpočtu. Rozhodl jsem se pro použití knihovny OpemCV, která je volně dostupná jako open source a kterou jsem použil i při implementaci softwarového rozhraní. Díky této knihovně se realizace stala velmi jednoduchá.

Pro měření času v obou aplikacích jsem použil funkci `clock()` z knihovny time.h. Funkce neměří čas, ale počítá takty, a proto je na konci rozdíl taktů podělen počtem taktů za sekundu. Pomocí této funkce dokáži změřit čas s přesností na milisekundy. Protože přesnost není příliš velká, vybral jsem pro testování obrazy s větším rozlišením. Měřil jsem pouze délku samotné filtrace a dobu ukládání dat z PC do paměti na Pico kartě a zpět. Časy načítání a ukládání dat z/do souboru jsem neměřil, protože jsou u obou aplikací shodné.

U hardwarové řešení znám frekvenci, na které vybraná platforma pracuje, a tak si může celkový čas filtrace spočítat. Každý takt je na výstupu z filtru jeden výsledný bod obrazu. První platný výstup má zpoždění, které se rovná délce jednoho řádku obrazu plus zpoždění filtru, které je v tomto případě zanedbatelné. Dále nesmíme zapomenout, že délka řádku je o dva body delší než původní obraz, protože se replikují krajní pixely. Celkový čas filtrace t se tedy spočítá

$$t = \frac{(w+2)\times(h+1)}{f} \tag{6.1}$$

kde w je šířka obrazu, h je výška obrazu a f je frekvence, která má u zvolené platformy hodnotu 66 MHz. Při tom se ještě zanedbává čas ukládání posledních bodů obrazu po dokončení filtrace.

Naměřené a spočítané výsledky jsou uvedeny v tabulce 6.1. Každé měření bylo provedeno desetkrát a hodnoty byly zprůměrovány. Softwarová aplikace běžela na počítači s procesorem AMD Athlon 64 X2 Dual-Core s frekvencí 1,7 GHz, pamětí 1 GB 667 MHz a 32-bitovým operačním systémem Windows Vista.

Rozlišení obrazu	Software	Hardware			
	t_F	t_F	t_T	t_W	t_R
1000×1000	0,079	0,015	0,015	0,441	1,131
2000×1000	0,156	0,030	0,030	0,868	2,248
2000×2000	0,311	0,060	0,061	1,736	4,523
3000×2000	0,465	0,091	0,091	2,621	6,828
3000×3000	0,716	0,136	0,137	3.888	10,148
4000×3000	0,961	0,181	0,182	5,177	13,769
4000×4000	1,274	0,241	0,243	6,903	18,198

Tabulka 6.1: Naměřené časy filtrace obrazu

t_F doba trvání filtrace
t_W doba trvání zápisu dat do paměti na Pico kartě
t_R doba trvání čtení dat z paměti na Pico kartě
t_T teoretická doba trvání filtrace, spočteno podle vztahu 6.1

Z výsledků můžeme vidět, že hardwarová implementace filtru je zhruba pětkrát rychlejší než efektivní softwarová implementace. Pokud bychom však brali v úvahu i přenos dat z/do FPGA, dosahuje softwarová aplikace mnohem lepších výsledků, neboť přenos dat z PC do paměti připojené k FPGA dosahuje mnohem nižší propustnosti v porovnání s propustností mezi procesorem a pamětí RAM. Doby těchto operací lze také vidět v tabulce 6.1. V praktické realizaci by se proto navržený filtr nejlépe uplatnil, kdyby byl zakomponován do nějaké hardwarové architektury.

7 Závěr

V této práci jsem se věnoval metodám hardwarové akcelerace filtrace obrazu. Seznámil jsem se s principem filtrace obrazu a s různými technikami akcelerace, které se používají při realizace pomocí hradlových polích FPGA. Praktickou část práce pak tvořila implementace obecného lineárního obrazového filtru metodou distribuované aritmetiky v jazyce VHDL a implementace softwarového rozhraní v jazyce C++. Byla navržena architektura, která umožňuje realizovat libovolný lineární filtr pracující s maskou o velikosti 3×3.

Navržené řešení bylo důkladně otestováno a bylo porovnáno s optimalizovaným čistě softwarovým řešením využívající instrukce MMX a SSE. Experimentálně i výpočtem bylo ukázáno, že rychlost filtrace v hardwaru je několikanásobně větší, což bylo cílem této práce. I s mnohem nižším taktovacím kmitočtem (66 MHz) bylo dosaženo pětinásobného urychlení v porovnání s řešením běžícím na CPU 1,7 GHz a 667 MHz RAM paměti.

Práce měla pro mě velký přínos po stránce teoretické, ale hlavně praktické. Seznámil jsem se podrobněji s jazykem VHDL a vytvářením aplikací pro FPGA. Kromě toho jsem zjistil, že ladění hardwarových aplikací může být někdy obtížné, protože není dostupný žádný debugger. Toto jsem si zkusil hlavně při implementaci řadiče paměti, protože jsem si nemohl hardwarovou paměť odsimulovat v prostředí ModelSimu.

Navržená architektura by se mohla dále rozšířit. Vhodným rozšířením by bylo zvětšení velikosti filtrační masky z 3×3 na 5×5 nebo dokonce na 7×7. I když bylo věnováno nemalé úsilí optimalizaci z hlediska maximálního výkonu, určitě existuje řada možností, jak navržené řešení vylepšit.

Co se týče výhledu do budoucna, tak bych se dále zaměřil na realizaci složitějších nelineárních filtrů, kde bych mohl využít získané znalosti z této práce. Filtry bych implementoval specializované pouze na jednu konkrétní operaci, což je více typičtější pro hardware.

Literatura

[1] Meyer-Baese, U.: *Digital signal processing with field programmable gate arrays*. Berlin, New York: Springer, 2007, třetí vydání, ISBN 978-3-540-72612-8.

[2] Kršek, P.: *Základy počítačové grafiky: IZG*. Brno: FIT VUT v Brně, 2008.

[3] Lichý Stanislav: *Techniky používané pro ostření a rozmazávání obrazu*. Brno, 2008, bakalářská práce, FIT VUT v Brně.

[4] Pikora, J.: *Implementace grafických filtrů pro zpracování rastrového obrazu*. Brno, 2008, bakalářská práce, Masarykova universita, Fakulta informatiky.

[5] Čapka, L.: *Akcelerace grafických operací s využitím FPGA*, diplomová práce, Brno, FIT VUT v Brne, 2007.

[6] *E-16 Hardware technical reference*. [online], 2006, [cit. květen 2009]. URL http://www.picocomputing.com/pdf/www/E-16/E-16%20Hardware%20Reference%20Manual/E16%20Rev%20D%20Hardware%20Guide.pdf

[7] *PSRAM MT45W8MW16BGX*. [online], 2004, [cit. květen 2009]. URL http://www.picocomputing.com/pdf/www/E-16/Third%20Party%20Manuals/PSRAM/MT45W8MW16BGX.pdf

[8] Čerba, O.: *Barevné modely*. [online], [cit. květen 2009]. URL http://gis.zcu.cz/studium/pok/Materialy/Book/ar03s01.html

[9] *Metody zvýrazňování obrazu 2: Prostorová zvýraznění - filtrace*. [online], [cit. Květen 2009]. URL http://geogr.muni.cz/archiv/vyuka/DPZ_CVICENI/Texty/DZO_05_zvyrazneni_2.pdf

[10] Fisher, R., Perkins, S., aj.: *Mean Filter*. [online], 2003, [cit. květen 2009]. URL http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm

[11] Fisher, R., Perkins, S., aj.: *Gaussian Smoothing*. [online], 2003, [cit. květen 2009]. URL http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[12] *FPGA implementation of FIR filter*. [online], [cit. květen 2009]. URL http://asic-soc.blogspot.com/2008/06/fpga-implementation-of-fir-filter.html

[13] Pech, J.: *Nebojte se FPGA*. [online], 2009, [cit. květen 2009]. URL http://hw.cz/Teorie-a-praxe/Dokumentace/ART365-Nebojte-se-FPGA.html

[14] *Dynamic random access memory*. [online], 2009, [cit. květen 2009]. URL http://en.wikipedia.org/wiki/Dynamic_random_access_memory

Seznam příloh

Příloha 1. CD se zdrojovými texty, manuálem k vytvořené aplikace a touto zprávou v elektronické podobě